```python
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 28 15:38:36 2019

@author: Nate
"""


import numpy as np
import matplotlib.pyplot as plt
import pdb

'''
Use the Killingbeck method as presented in class to solve for the eigenvalues of the
hydrogen atom as in 1), but with greater accuaracy. Use the same Verlet algorithm
to integrate backward to the origin from r = 25-50. Do Newton’s iterations 4-10
times to find the correct e so that u(0,e) = 0. Determine the lowest energy levels of
l = 0, 1, 2, 3. Use e = -0.6 as your initial guess energy. Plot all energy values as a
function of dr from 0.01 to 0.1.
'''

delta_r = []
l_orbital = [0,1,2,3]
#l_orbital = [0]
energy_guess = -.6

#-1/r potential energies
for i in range(1,11):
#for i in range(1,50):
    delta_r.append(i*.01)

#u(r+dr), u(r), l, r, energy_guess
def stepping(u_r_plus_dr, u_r, l, r, eps):
    #fun = 2*(r*r/2) + l*(l+1)/r**2 - 2*eps
    #pdb.set_trace()
    fun = -2/r + l*(l+1)/r**2 - 2*eps
    u_r_minus_dr = 2*u_r - u_r_plus_dr + delt_r**2*fun*u_r
    return(u_r_minus_dr)

def stepping_for_v(v_r_plus_dr, v_r, u_r, l, r, eps):
    #fun = 2*(r*r/2) + l*(l+1)/r**2 - 2*eps
    #pdb.set_trace()
    fun = -2/r + l*(l+1)/r**2 - 2*eps
    v_r_minus_dr = 2*v_r - v_r_plus_dr + delt_r**2*fun*v_r+ delt_r**2*-2*u_r
    return(v_r_minus_dr)


##############################################################
##############################################################


good_points = []
to_plot = []


for l in l_orbital:
    to_plot_holder = []
    for delt_r in delta_r:

        r = 100 - delt_r*2

        for it in range(20):  #8
            u = [0, .01]  #u_r_plus_dr, u_r
            v = [0, .01]
            #print(int(round(r/delt_r)))
            
            
```

```python
            for i in range(int(round(r/delt_r))):
                
                u[0],u[1] = u[1], stepping(u[-2],u[-1], l, r, energy_guess)
                v[0],v[1] = v[1], stepping_for_v(v[-2],v[-1], u[0], l, r, energy_guess)
                #u.append(stepping(u[-2],u[-1], l, r, energy_guess))
            
                if u[0]*u[1] < 0:
                    good_points.append([l, energy_guess, r])

                r-=delt_r
                #u = [u[-2],u[-1]]
            energy_guess = energy_guess - u[0]/(v[0]+.000001)

#        print("delta r is: ", delt_r, "  Energy is: ",energy_guess)
        to_plot_holder.append([delt_r, energy_guess])
    to_plot.append(to_plot_holder)


###############################################################
###############################################################
#Plotting


l0 = []
l1 = []
l2 = []
l3 = []

for i in range(len(good_points)):
    if good_points[i][0] == 0:
        l0.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 1:
        l1.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 2:
        l2.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 3:
        l3.append([good_points[i][1],good_points[i][2]])


fig1, axes1 = plt.subplots()

axes1.scatter([to_plot[0][i][0] for i in range(len(to_plot[0]))], [to_plot[0][i][1] for i in range(len(to_plot[0]))])
axes1.scatter([to_plot[1][i][0] for i in range(len(to_plot[1]))], [to_plot[1][i][1] for i in range(len(to_plot[1]))])
axes1.scatter([to_plot[2][i][0] for i in range(len(to_plot[2]))], [to_plot[2][i][1] for i in range(len(to_plot[2]))])
axes1.scatter([to_plot[3][i][0] for i in range(len(to_plot[3]))], [to_plot[3][i][1] for i in range(len(to_plot[3]))])
axes1.set_ylabel('Energy')
axes1.set_xlabel('$\Delta$r')
axes1.set_title("Energy as a Function of $\Delta$r", va='bottom')
axes1.legend(('l=0','l=1','l=2', 'l=3'), loc='upper right')
#plt.show()
```